第3章

ARMプロセッサの上でリアルタイムOSを動かす

――本誌2006年3月号付属ARM7基板上にTOPPERS /JSP 1.4.2を移植

中野俊和，平尾智也，赤星博輝

ここでは，ARMプロセッサをターゲットに，リアルタイムOSを移植した事例を紹介する．移植したリアルタイムOSはμITRON 4.0規格に準拠した「TOPPERS /JSP 1.4.2」，ターゲットは本誌2006年3月号に付属したCPU基板上のARMプロセッサである．TOPPERSはソース･コードが公開されている．実際のリアルタイムOSのコードを見ることは，貴重な経験になるだろう．なお，本記事で紹介するプログラムやファイルは，本誌付属のCD-ROMに収録されている．　（編集部）

高機能化している家電製品（エアコン，炊飯器，テレビなど），携帯電話，工業用ロボット，そして自動車というと，あまりつながりがないように見えますが，それぞれコンピュータで制御されているという共通点があります．このような特定の機器に対して組み込まれるコンピュータ・システムを「組み込みシステム」，そしてその上で動作するソフトウェアを「組み込みソフトウェア」と呼びます．

● マルチタスクにはプログラム間通信や割り込みが必要

組み込みソフトウェアで実現する機能を，携帯電話を例に考えてみましょう．最近の携帯電話を見ると，さまざまな機能が実現されています．「電話をかける」，「アドレス帳を管理する」，「メールを読み書きする」，「写真を撮る」，「音楽を聴く」，「予定表を管理する」など，複数のしごとを実行することができます．このように複数のしごと（タスク）を同時に（厳密には同時ではないが…）実行することを「マルチタスク」といいます．

次に，このマルチタスクを実現するための機能を考えてみます．まず，複数のタスクを実行したときに，タスク間でデータのやり取りをしたい場合があります．写真を撮って，その画像をメールで送信する場合などです．その場合には，プログラム間で通信するためのしくみが必要です．また，メールを書いているときに電話が鳴った場合，電話に出て，また続きを書くといったように，いったん処理を停止し，別の処理を実行するという，割り込みの機能が必要です．

さらに，このような機能を実現するだけでなく，時間に関するルールを守る必要があります．例えば，メールを書いているときに電話が鳴った場合，通話するまでに通話ボタンを押して30秒も待たされたら，多くのユーザは怒り出すことになります．人間に対するインターフェースでは，応答時間は数十ms（ミリ秒）あれば十分なことも多いのですが，ハードウェア相手の場合は，もっと早い応答が求められることもあります．

● リアルタイムOSは組み込みシステムの基盤ソフト

マルチタスク・システムを動作させるための基盤ソフトウェアを一から作っていくのはたいへんな作業になります．ここで，組み込みソフトウェアの基本となる機能を提供するのがリアルタイムOS（RTOS）です．リアルタイムOSを使うことで，基本となる部品のうえにアプリケーションを実現することができるため，開発期間の短縮や再利用性の向上が可能となります．リアルタイムOSはマルチタスク機構，タスク間同期・通信，割り込み管理，例外管理，時間管理，メモリ管理といったしくみを提供するので，開発

KeyWord　TOPPERS，マルチタスク，リアルタイムOS，ポーティング，割り込み動作，ARM7，スタートアップ・ルーチン，タイマ，UART，カーネル



者はこのしくみのうえにさまざまな機能を実現することができます[1]．

では，リアルタイムOSにはどのようなものがあるのでしょうか？ 代表的なものとして，米国WindRiver社の「VxWorks」，自動車などで用いられるOSEK準拠のOS，Linuxにリアルタイム機能を追加したもの，トロンプロジェクトの一環として標準化されたμITRON準拠のOS[2]などがあります．μITRON準拠のOSは，日本国内で組み込みOSとして高いシェアを得ています．今回，本誌2006年3月号の付属ARM7ボードに移植したのは，このμITRON 4.0規格に準拠した「TOPPERS /JSP 1.4.2」です．

● TOPPERSプロジェクト，TOPPERS/JSPとは？

TOPPERSプロジェクト[3]は，ITRON仕様の技術開発成果を出発点とし，組み込みシステム構築の基盤となる各種のソフトウェアを開発し，良質なオープン・ソース・ソフトウェアとして公開することで，組み込みシステム技術と産業の振興を図ることを目的としたプロジェクトです．2003年9月に設立された特定非営利活動法人（NPO法人）を中心に，名古屋大学の高田広章氏をリーダとして活動しています注．

TOPPERSのおもしろい部分としてライセンスがあります．**図1**のように“ TOPPERSライセンス ”と呼ぶ独自の利用条件が設定されています．GNU GPLなどの既存の利用条件を適用するのではなく，独自の利用条件になっています．

今回，ARM7基板に移植したのはTOPPERS/JSP（バージョン1.4.2）になります．JSPとは，Just Standard Profileの略で，μITRON 4.0仕様のスタンダード・プロファイル規定に従って実装されています．このJSPカーネルには，以下のような特徴があります．

- 読みやすく改造しやすいソース・コード
- ほかのターゲットCPUへのポーティング（移植）が容易な構造
- 高い実行性能と小さいRAM使用量
- Linux上およびWindows上でのシミュレーション環境
- 開発環境まで含めてフリー・ソフトウェアのみで構築可能

● OSの作成は難しいが，移植はそれほど難しくない

リアルタイムOSは，ソフトウェアとしては高度なプログラムと言ってよいと思います．理由としては，1）割り込みや例外処理により複数のタスクを切り替えるため，動作が複雑，2）その割り込みを制御するためにアセンブラによるプログラミングが必須，という点が挙げられます．

まず，割り込みにおいてどのような処理が行われているかを**図2**で見てみます．最初，プログラムAを実行していて100番地の命令を実行しようとしたときに割り込みがかかったとします．そうすると，まずCPUの中にあるデータをメモリに書き出します（そうしないと，別のプログラムの処理を開始した場合，これまでの結果が上書きされてしまう）．それで，CPUでは戻り番地を覚えておき，プログラムBの実行が終わったら，先ほどメモリに書き出したデータをCPUに戻し，プログラム・カウンタ（PC）をリターン・アドレスの内容に変更することで，プログラムAを再実行することができます．

このようなCPUの内部の情報にアクセスするためには，アセンブリ言語を使ったプログラミングを行う必要があり

Copyright（C）<開発年> by <著作権者1>
Copyright（C）<開発年> by <著作権者2>
...

上記著作権者は，以下の（1）～（4）の条件か，Free Software Foundationによって公表されているGNU General Public LicenseのVersion 2に記述されている条件を満たす場合に限り，本ソフトウェア（本ソフトウェアを改変したものを含む．以下同じ）を使用・複製・改変・再配布（以下，利用と呼ぶ）することを無償で許諾する．

（1）本ソフトウェアをソースコードの形で利用する場合には，上記の著作権表示，この利用条件および下記の無保証規定が，そのままの形でソースコード中に含まれていること．

（2）本ソフトウェアを，ライブラリ形式など，他のソフトウェア開発に使用できる形で再配布する場合には，再配布に伴うドキュメント（利用者マニュアルなど）に，上記の著作権表示，この利用条件および下記の無保証規定を掲載すること．

（3）本ソフトウェアを，機器に組み込むなど，他のソフトウェア開発に使用できない形で再配布する場合には，次のいずれかの条件を満たすこと．

（a）再配布に伴うドキュメント（利用者マニュアルなど）に，上記の著作権表示，この利用条件および下記の無保証規定を掲載すること．

（b）再配布の形態を，別に定める方法によって，TOPPERSプロジェクトに報告すること．

（4）本ソフトウェアの利用により直接的または間接的に生じるいかなる損害からも，上記著作権者およびTOPPERSプロジェクトを免責すること．

本ソフトウェアは，無保証で提供されているものである．上記著作権者およびTOPPERSプロジェクトは，本ソフトウェアに関して，その適用可能性も含めて，いかなる保証も行わない．また，本ソフトウェアの利用により直接的または間接的に生じたいかなる損害に関しても，その責任を負わない．

図1　TOPPERSライセンス全文

TOPPERSライセンスの中でもっとも特徴的なのは，条件（3）の（b）である．

注：今回のARM7基板へのTOPPERS/JSPカーネルの移植に関して，名古屋大学の高田・冨山研究室の方々にご指導，ご協力いただきました．心より御礼申し上げます．

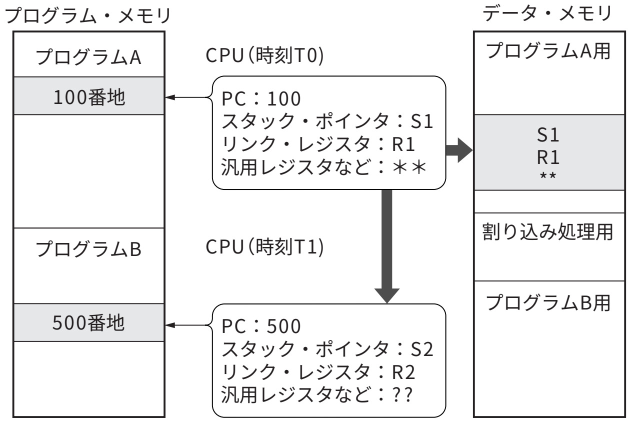


図2　割り込み動作

時刻T0でプログラムAを実行していたときに割り込みがかかり，プログラムBを実行することになった場合，CPUの内容をスタックに書き出し，プログラムBの実行を開始する．プログラムBの実行が完了したら，メモリに書き出した内容をCPUに戻し，PCを100に戻す．

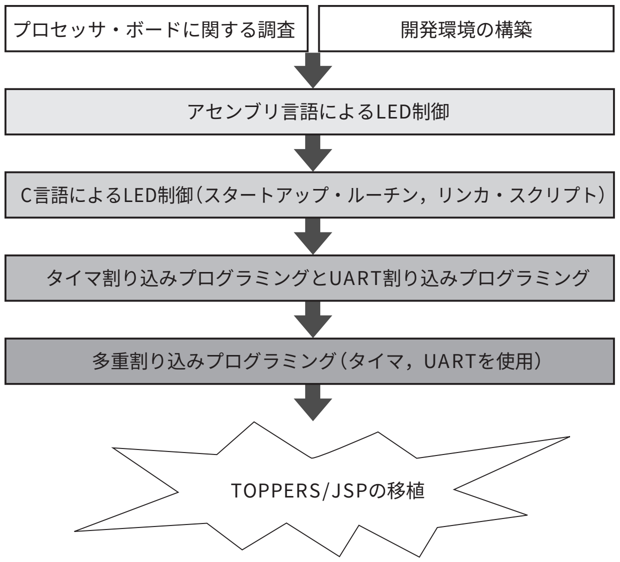


図3　移植フロー

RTOSをいきなり移植するのはたいへんである．ボードを理解するところから，徐々に準備していく．

表1　ARMの実行モード

七つあるが，TOPPERS/JSPではスーパバイザ・モード(SVC)と割り込みモード(IRQ)の二つを使用する．タスクもSVCで動作させる．

	モード	内　容
USR	ユーザ・モード	一般的なプログラムが実行する
FIQ	高速割り込みモード	IRQより高速な割り込み処理を行う
IRQ	割り込みモード	汎用割り込みの処理を行う
SVC	スーパバイザ・モード	OSの保護モードで使用する
ABT	アボート・モード	データ読み出し，命令のプリフェッチに失敗した場合のモード
SYS	システム・モード	OSの特権ユーザ・モード
UND	未定義命令モード	未定義命令を実行した場合のモード

ます．C言語では，どのレジスタにどの値を入れるかを指定することができませんし，制御レジスタの内容を変更したり，割り込みを禁止したりといった記述を書けません．例えば，今回使用するARMプロセッサにおける内部の情報の書き出し・復帰には，以下のような記述を使っています．

```
stmfd sp!, {r4 - r11,lr}
        /* レジスタR4～R11, LRのスタックへの保存 */
ldmfd sp!, {r4 - r11,lr}
        /* スタックからレジスタR4～R11, LRの復帰 */
mrs   r0, cpsr
                /* ステータス・レジスタをR0に代入 */
msr   cpsr, r0
         /* R0の値をステータス・レジスタに代入 */
```

複雑に動くタスクを制御するリアルタイムOSの作成はたいへんですが，リアルタイムOSの移植となると話は変わってきます．似た環境に対してすでに実装したものがあれば，比較的に簡単に移植することができるのです．

以下では，実際の移植作業について説明していきます．今回の移植は，大まかに次のような手順で行いました(**図3**)．

1) ターゲットCPUに関する調査
2) 簡単なプログラムの実行
3) 割り込みがあるプログラムの実行
4) TOPPERS/JSPカーネルの移植

● ターゲットCPUや開発環境の情報を収集する

まず，ターゲットとなるCPU，ARM7基板，コンパイラなどの開発環境について調査します．CPUは，「ARM7TDMI」というARMプロセッサ(ARMコア)です．このARM7TDMIは，ゲームボーイアドバンスなどの携帯型ゲーム機でも使われているプロセッサです．

ARMプロセッサには**表1**に示す七つの実行モードがあり，用途に応じて切り替わります．今回のTOPPERS/JSPでは，SVCモードとIRQモードのみを使用します．



ARM7TDMIのレジスタ構成は，汎用レジスタ`r0`～`r15`と`cpsr`というステータス・レジスタからなります．16本が汎用レジスタということになっていますが，すべてをプログラムに使えるわけではありません．`r15`はプログラム・カウンタとして使用します．

さらに，C言語とアセンブリ言語の両方を使ってプログラムを作成する場合，いくつかのレジスタはコンパイラが使用法を決めるため，勝手に使うことができなくなります．今回使用するコンパイラでは，`r13`はスタック・ポインタ，`r14`はリンク・レジスタ，`r0`～`r3`と`r12`はスクラッチ・レジスタ，`r4`～`r11`はバリアブル(variable)レジスタとなります．これを，コンパイラ・コンベンションと呼んでいます．TOPPERS/JSPではC言語とアセンブリ言語のプログラムが混在するので，このコンベンションに従ってプログラムを作成する必要があります．**図4**のように，`r13`と`r14`はモードによって切り替わることになっているので，注意が必要です．

このスクラッチ・レジスタとは，関数呼び出しを行って戻ってきた場合に，呼び出し前と後で値が変わっても問題のないレジスタです．一方，バリアブル・レジスタは，呼び出し前と後で値が変わってはいけないレジスタで，それぞれ役割が決められています．

使用するメモリ・マップを**図5**に示します．SRAM領域が8Kバイト，Flash/EE(フラッシュ・メモリ)領域が64Kバイトあります．基本的な使いかたとしてFlash/EE領域にプログラムを置き，スタックやグローバル変数などをSRAM領域に確保します．

● 開発環境にはGNUソフトウェアを利用

開発環境ですが，ARM7基板に付属のツールで使用したのは，「ARMWSD」というシリアル・ポート経由でフラッシュ・メモリへデータを書き込むソフトウェアです．そのほかの開発ツールとして，アセンブラ，コンパイラ，デバッガが必要になります．個別に集めるとめんどうなので，今回は「GNUWing 20030801 for arm-elf」を使用しました．GNUWingはアップウィンドテクノロジー・インコーポレイテッドが開発した組み込みシステム開発のためのGNUソフトウェアのディストリビューションで，今回使用したバージョンにはGCC(3.3)，Binutils(2.14)，GDB(5.3)，newlib(1.11.0)などが含まれています．このGNUWingのFTP版は，Webサイトで無償配布されています[(4)]．

デバッグ環境は，英国ARM社のMulti-ICEとMulti-ICE-Serverを名古屋大学の高田・冨山研究室のご協力によ

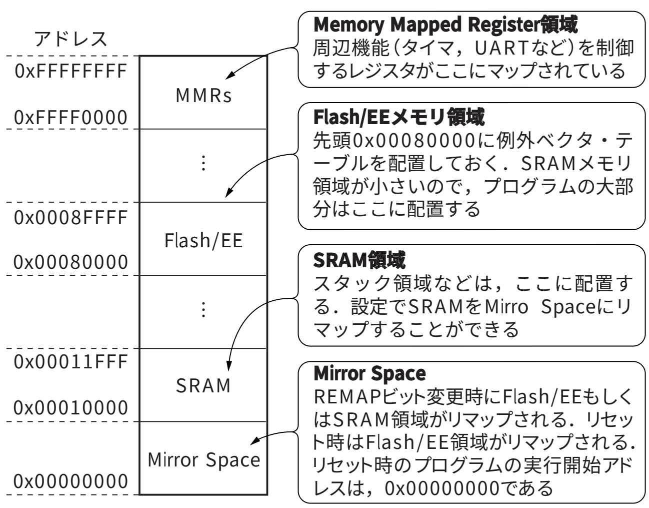


図5 ADuC7026チップのメモリマップ

メモリのどこに何を置くかは設計者が決める必要がある．基本的には，データはSRAM領域，プログラムはFlash/EE領域に配置するが，今回の移植では，Mirror Spaceの扱いでトラブルが発生した．

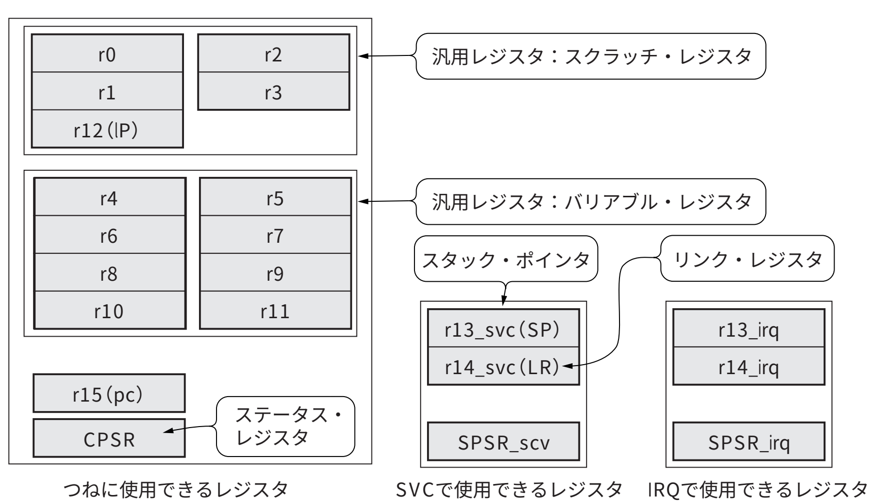


図4
ARM7TDMIのレジスタ構成

今回移植するTOPPERS/JSPは二つのモードを使用するが，そこで使用できるレジスタはモードによって異なる．

り使用させていただきました．Multi-ICE-GDB-Serverを使用し，GDBでデバッグを行います．Multi-ICE-GDB-Serverには，Cygwinのバージョンが新しいと動作しないという問題があったため，2003年ごろにリリースされたCygwin（DLLバージョン1.3.22-dontuse-21）をインストールして使用しました．

GDBの接続手順としては，Multi-ICEでボードのJTAG端子とパソコンを接続し，Multi-ICE-Serverを立ち上げます．Multi-ICE-ServerでARMコアを認識できたらMulti-ICE-GDB-Serverを起動し，GDB（Insight）をremoteで接続します．これでデバッグが可能となります．

● LED制御のアセンブリ・プログラムを作成する

移植する前にARM7基板の動作確認を行うため，基板に付いているLEDの点灯・消灯を繰り返すプログラムをアセンブリ言語で作成しました．今回は，CPUのチップそのものが40本のGPIO（general purpose bi-directional I/O）ピンを持っており，その先にLEDが接続されています．ソフトウェアの処理としては，GPIOについて設定した後，LEDを消灯→時間を稼ぐためのループ→LEDを点灯→ループ→LEDを消灯→ループ→LEDを点灯，…，という処理を繰り返します（**リスト1**）．

WindowsやUNIXのプログラム開発の場合，このような動作をC言語やアセンブリ言語で記述したら，あとはコンパイルするだけで終わりです．しかし，このようなボードで動作させるためには初期化を行う必要があります．このため，**リスト2**に示すスタートアップ・ルーチンを準備し，割り込みベクタの設定，およびリセット時に最初に呼び出す処理を指定します．

ここまでできれば，以下のコマンドでstartup.o，led_ctrl.oのオブジェクトを生成します．

```
> arm-elf-gcc -g -c -mlittle-endian startup.S led_ctrl.S
```

さらに，メモリ上のどこにプログラムやデータを配置するのかを決定する必要があります．このため，実行形式を作るときに，リンカ・スクリプト（linker_scr）で配置場所を指定しなければなりません．プログラムは基本的にはFlash/EE領域，データはSRAM領域という指定を行っています．以下のコマンドで，実行形式のled_ctrl.elfを作成しました．

```
> arm-elf-ld -g -EL -Tlinker_scr startup.o led_ctrl.o -o
  led_ctrl.elf -I. -L.
```

しかし，ここで生成された実行形式はELFフォーマットなのですが，ボードにダウンロードするためにはIntel Hexフォーマットに変換する必要があります．そのため，以下のコマンドを実行し，書き込み可能な実行ファイル（led_ctrl.hex）を作成します．

```
> arm-elf-objcopy -O ihex -S led_ctrl.elf led_ctrl.hex
```

3

リスト1 LEDを点滅するアセンブリ・プログラム（led_ctrl.S）

```
led_ctrl:
        ldr r0,  gpio_baseaddr      /* GPIO Registers Base Address */
        add      r1,  r0, #GP4CON   /* GP4CON -> r1    */
        add      r2,  r0, #GP4DAT   /* GP4DAT -> r2    */
        add      r3,  r0, #GP4SET   /* GP4SET -> r3    */
        add      r4,  r0, #GP4CLR   /* GP4CLR -> r4    */
        mov      r5,  #0x00000000   /*        */
        str      r5,  [r1]          /* GPIO port4 GPIO mode    */
        mov r5,  #P4_7_DIR
        str      r5,  [r2]          /* GPIO port4 pin7 output mode */
        mov      r6,  #LED_ON
led_off:
        mov      r7,  #LOOP         /* OFF loop */
        str      r6,  [r3]          /* LED OFF */
loop_1:
        sub      r7,  r7,  #1
        cmp      r7,  #0
        bne      loop_1
led_on:
        mov      r8,  #LOOP         /* ON loop */
        str      r6,  [r4]          /* LED ON */
loop_2:
        sub      r8,  r8,  #1
        cmp      r8,  #0
        bne      loop_2
        b        led_on
         nop
```

（注釈：アドレスの設定／GPIOのモード設定／I/Oの向きを設定／LEDをOFF／LEDをON）

リスト2　もっとも簡単なスタートアップ・ルーチン(startup.S)

リセットがかかると，割り込みベクタによりラベル`reset_vector`で指定されるラベルにジャンプする．ここでは`start`でラベル・ジャンプし，さらに`led_ctrl`にジャンプする．

```
vector_table:
      ldr       pc,   reset_vector
      ldr       pc,   undef_vector
      ldr       pc,   swi_vector
      ldr       pc,   prefech_vector
      ldr       pc,   data_abort_vector
       nop
      ldr       pc,   irq_vector
      ldr       pc,   fiq_vector

reset_vector:
      .long start
/*省略*/
      .text
      .align 2
      .globl start
start:
      b         led_ctrl
       nop
```


あとは，でき上がったファイルをARMWSDというツールを使って書き込むだけです．また，実際にはこの一連の作業を行うのにMakefileを使用し，makeコマンドを実行するだけで，実行ファイルができ上がるようにしています．

● LED制御のC言語プログラムを作成する

次に，C言語を使ってLED制御を行います．ここではコンパイラのコンベンションの理解，スタートアップ・ルーチンの作成，リンカ・スクリプトのブラッシュアップを行い，TOPPERS/JSPの移植のために使うパーツをそろえていきます．

まず，スタートアップ・ルーチンとリンカ・スクリプトですが，TOPPERS/JSPカーネルのconfig/armv4/integratorを参考に作成し，例外のジャンプ先などを変更しています．

led_ctrl.cの内容は，アセンブリのLED制御とほぼ同じですが，`if`文でLEDの状態を見て，消灯していれば点灯，点灯していれば消灯としました(**リスト3**)．今回はC言語を使用するため，スタートアップ・ルーチン(startup.S)でスタック・ポインタの設定，動作モードの決定(スーパバイザ・モード．以下，SVCモード)，グローバル変数を格納するデータ領域の初期化などを行います．

今回は，コンパイルからリンクまでを一気に実行します．ただし，startup.Sは変更がないかぎりアセンブリを行う必要がないので，startup.oを最初に用意しておきました．以下のコマンドでコンパイルします．

```
> arm-elf-gcc -O2 -g -mlittle-endian -mcpu=arm7tdmi
  -nodefaultlibs \
  -nostartfiles -Wl,-T -Wl,linker_scr -Wl,-EL startup.o
  led_ctrl.c  \
  -o led_ctrl.elf -I. -L.
```

上のコマンドを実行することで，led_ctrl.elfができ上がります．このファイルから先ほどと同じように`arm-elf-objcopy`を使い，Intel Hexフォーマットのファイルを作成すれば，ボードにダウンロードできます．

● タイマ割り込みのプログラムを作成する

TOPPERS/JSPカーネルを移植するとき必要になるの

リスト3
LEDを点滅するCプログラム
(led_ctrl.c)

```
#include" led_ctrl.h"
#include" arm7tdmi.h"

typedef         unsigned long U32B;

void    led_ctrl(void)  {
    U32B gp4con_addr = GPIO_BASE + GP4CON;
    U32B gp4dat_addr = GPIO_BASE + GP4DAT;
    U32B gp4set_addr = GPIO_BASE + GP4SET;
    U32B gp4clr_addr = GPIO_BASE + GP4CLR;
    int i, j;

    *((volatile U32B *) gp4con_addr) = (U32B)0x00;
    *((volatile U32B *) gp4dat_addr) = P4_7_DIR;
    while(1) {
        if((*((volatile U32B *) gp4dat_addr)) & LED_ON) {
            *((volatile U32B *) gp4clr_addr) = LED_ON;
        } else {
            *((volatile U32B *) gp4set_addr) = LED_ON;
        }
        for(i=0; i<LOOP; i++) ;
    }
}
```

リスト4 タイマの設定

```
hardware_init_hook:
        /* PLLCON setting */
        ldr      r0,  =PLL_BASE
        mov      r1,  #PLLKEY1_DATA
        str      r1,  [r0,#PLLKEY1_VCT]
        mov      r1,  #(PLLCON_MDCLK_PLL|PLLCON_OSEL)
        str      r1,  [r0,#PLLCON_VCT]
        mov      r1,  #PLLKEY2_DATA
        str      r1,  [r0,#PLLKEY2_VCT]
        /* POWCON setting */
        mov      r1,  #POWKEY1_DATA
        str      r1,  [r0,#POWKEY1_VCT]
        mov      r1,  #(POWCON_PC_ACT|POWCON_CD_41M)
        str      r1,  [r0,#POWCON_VCT]
        mov      r1,  #POWKEY2_DATA
        str      r1,  [r0,#POWKEY2_VCT]
        mov      pc,  lr
        nop
```

PLLCONを書き換えるために必要な手順

内部発振器を使用，クロッキング・モードをPLL．デフォルトでこの設定

アクティブ・モード，CPUクロック41.78MHzに設定

POWCONを書き換えるために必要な手順

リスト5 割り込み時の動作

IRQによる割り込みが発生すると，vector_tableのirq_vectorで指定されたラベルにジャンプし，そこからCプログラムのtimer1_int_handlerを呼び出す．

```
vector_table:
     ldr        pc,  reset_vector
     ldr        pc,  undef_vector
     ldr        pc,  swi_vector
     ldr        pc,  prefech_vector
     ldr        pc,  data_vector
    nop
     ldr        pc,  irq_vector
     ldr        pc,  fiq_vector

irq_vector:
       .long interrupt

interrupt:
      stmfd      sp!,  {r0-r3,ip,lr,pc}  /* GPR -> stack */
      sub        r0,  lr,  #4          /* return address -> r0 */
      str        r0,  [sp,#0x18] /* over-write pc (<- r0) */
      mrs        r1,  spsr_all
      stmfd      sp!,  {r1}           /* r0(SPSR_IRQ)->stack */

      bl timer1_int_handler
```

IRQ割り込み発生

（a）アセンブリ言語

```
void  timer_int(void)  {
  unsigned long t1_con;
  volatile int i = 1;

  led_init();
  timer1_init();
  int_init();
  ena_irq();
  while(i == 1) ;
}

void  timer1_int_handler(void)
{
    *((volatile unsigned char *)T1CLRI) = 1;
    if((*((volatile U32B *)GP4DAT)) & LED_ON) {
         *((volatile U32B *)GP4CLR) = LED_ON;
     } else {
         *((volatile U32B *)GP4SET) = LED_ON;
     }
}
```

通常時はこのループが回っている

（b）Cプログラム

が，タイマ割り込みとUART割り込みです．この二つを単独で動作させ，その後，両方を使って多重割り込みを行い，最終的なTOPPERS/JSPの移植を行います．

まず，タイマ割り込みでは，1秒ごとにタイマ割り込みを発生させ，LEDを点灯，消灯させることにします．前述のLED制御を，ループによる時間を調整せずにタイマ割り込みで実現してみます．

スタートアップ・ルーチンは，C言語のLED制御とほぼ同じですが，割り込みを使用するとIRQモードでプログラムを実行するので，IRQモードのスタック・ポインタの設定を行っています．リンカ・スクリプトでは，このためにSVCモードのスタック領域(__stack)とIRQモードのスタック領域(__stack_int)を確保しています．

タイマを使うためには，そのタイマに使用するクロックを設定しなければなりません．この設定はスタートアップ・ルーチンにあるhardware_init_hook(**リスト4**)で，CPUクロックの周波数を41.78MHzでアクティブ・モードに設定しました．

リスト5(b)のメイン関数(timer_int)では，LEDの初期化，割り込みコントローラの初期化(タイマ1のIRQを有効に)，タイマの初期化(1秒周期のカウント値，ダウン・カウント，割り込み有効などの設定)，プロセッサのIRQ割り込みの許可を行っています．その後，メイン関数は無限ループに入り，とくに何も処理しません．i==1は割り込み処理でスタックが壊れていないかどうかをチェックするための記述です．

タイマ1が1s(秒)分カウントしたら(カウント値が0になったら)，割り込みが発生し，タイマ1割り込みが有効であれば，割り込みコントローラからプロセッサに割り込み信号が送られます．プロセッサでIRQを許可していれば，IRQ例外発生(IRQモード)となり，例外ベクタ・テーブルが示すラベルinterruptにジャンプします(**リスト5(a)**)．



リスト6 UARTの初期化

```
Inline void  uart_init(void)    {
     /* Port setting : UART */
     *((volatile U32B *)GP0CON) = GP07_SIN;
     *((volatile U32B *)GP1CON) = (GP10_SIN|GP11_SOUT|
                                    GP12_RTS|GP13_CTS|GP14_RI|
                                    GP15_DCD|GP16_DSR|GP17_DTR);
     *((volatile U32B *)GP2CON) = GP20_SOUT;
     /* Baud Rate = 9600 */
     *((volatile U32B *)COMCON0) = COMCON0_DLAB;
     *((volatile U32B *)COMDIV0) = COMDIV0_9600;
     *((volatile U32B *)COMDIV1) = COMDIV1_9600;

     /* Word length is 8 bits */
     /* Stop bit is 1 bit */
     *((volatile U32B *)COMCON0) = (COMCON0_WLS_8b | COMCON0_STOP);

     /* Enable “TX buffer empty"  and “RX buffer full"  interrupt */
     *((volatile U32B *)COMIEN0) = (COMIEN0_ETBEI | COMIEN0_ERBFI);
}
```

GPIOのモード設定

ボーレートを設定

データ長8ビット，ストップ・ビット1ビットを設定

割り込みの設定

このとき，割り込みが入る前はSVCモードで実行していますが，IRQ割り込みが入るとIRQモードに自動的に切り替わります．このとき，いくつかのレジスタもIRQ用に切り替わります．スタック・ポインタ(r13_irq)がIRQ用のスタック・エリアを指し，リンク・レジスタ(r14_irq)にはプリフェッチを中断された命令のアドレスが格納され，割り込み前のcpsrの値がSPSR_irqに自動で保存されます．

interruptにジャンプしたら，プログラム側でそれまで使用していたレジスタをスタックに保存します．保存するレジスタは全部ではなく，r0～r3，ip(r12)，lr(r14)，pc(r15)です．それ以外は，この割り込み出入り口処理で使用する場合に保存します．

割り込み入り口の処理を終えると，やっと割り込みハンドラ(timer1_int_handler)です．ここは，タイマ1割り込み要求をクリアし，LEDの制御を行います．割り込み要求のクリアですが，T1CLRIレジスタに何でもいいので値を書き込むことで実現できます．**リスト5(b)**では，1を書き込んでいます．

割り込みハンドラが終わると，割り込み出口処理としてスタックに保存したspsrレジスタの値を復帰し，spsrをcpsrにコピーすることで，割り込み前の状態に戻します．

リスト7 UARTのメイン処理と割り込み部

```
void  uart(void)  {
   volatile int i = 1;

    led_init();     /* LED initialize */
    timer1_init(); /* Timer1 initialize */
    int_init();     /* Interrupt Controller initialize */
    uart_init();    /* UART initialize */
    ena_irq();      /* IRQ enable */
    while(i == 1)  ;
}

void  uart_int_handler(void) {
    volatile U32B iid;
    U32B rdata;
    if((*((volatile U32B *)COMSTA0)) & COMSTA0_DR) {
        /* RX buffer is full */
        iid = *((volatile U32B *)COMIID0);
        /* Echo back */
        rdata = *((volatile U32B *)COMRX);
        *((volatile U32B *)COMTX) = rdata;
    } else {
          /*  省略 */
      }
}
```

通常時はこのループが回っている

割り込みが発生すると呼び出される

受信バッファが満杯の場合

受信したデータを送信する

● UART割り込みのプログラムを作成する

UART割り込みですが，タイマ割り込みと大部分は同じです．UARTによる割り込みが発生したら，uart_int_handlerを呼び出すようにします．また，タイマについては割り込みが発生しないように変更しました．

ここでは，UARTでエコーバックを行うプログラムを作成しました．パソコン側から送信したデータをそのまま返すというものです．パソコン側にTeratermなど，シリアル・ポートに接続できるソフトウェアを用意し，ボーレート9600bps，データ長8ビット，パリティなし，ストップ・ビット1ビット，フロー制御なしと設定することで，ARM7基板と通信することができます．

リスト6に示すように，UARTの初期化(uart_init)を行います．まず，シリアル通信ができるように設定を行い，次にボーレート，データ長，ストップ・ビットの設定を行っています．その後，どの状態で割り込みを発生させるかを設定します．今回は，データを送信して送信バッファが空になったときと受信バッファが満杯になったとき(受信バッファは1キャラクタ分しかないので，データを受信したときと同じ)に割り込みが発生するように設定します．

各初期化が終わり，IRQ割り込みを許可すると，メイン関数（uart）は無限ループに入ります．

リスト7の割り込みを待っているので，パソコン側からシリアル・ポート経由でデータを送信するまでループが回っていますが，パソコンのキーボードから例えば“a”を入力すると，“a”という文字がシリアル通信で送信され，受信バッファにデータが入ります．受信バッファにデータが入ると，IRQ割り込みが発生し，uart_int_handlerが呼び出されて処理を行います．

● 多重割り込みのプログラムを作成する

ここまできたら，タイマとUARTの二つの割り込みを受け付ける多重割り込みのプログラムを作成します．まず，多重割り込みを行うにあたって，割り込みの優先順位を決める必要があります．TOPPERS/JSPではタイマが優先されているので，ここでもタイマをUARTより優先することにします．UART割り込み中のタイマ割り込みは可能ですが，逆にタイマ割り込み中のUART割り込みは受け付けないようにします．これは，割り込みをマスクすることによって実現します．それから，割り込みの入れ子を表す変数nestを用意し，ほんとうに多重割り込みが成功しているかどうかを確認することにしました．

次に，割り込みハンドラの処理も決めます．UARTのハンドラでは，先ほどのエコーバックを実現します．タイマ1のハンドラではLEDの点滅を行い，入れ子の数（nest）をUARTで送信することにします．タイマ1は，1秒ごとに起動するようにしているため，1秒ごとにLEDの反転と入れ子の数の送信が行われます．

今回はここで苦労することになりました．今まで，割り込みはIRQモードで処理してきており，この多重割り込みもIRQモードで処理しようとしたのですが，ここで落とし穴にはまってしまいました．

通常の実行状態（SVCモード）でIRQ割り込みが入ると，sp_svc，lr_svcがIRQ用のsp_irq，lr_irqに切り替わり，cpsrがSPSR_irqに保存されます．このlr_irqには，割り込みが終了したときに戻るべきアドレスが入っています．しかし，割り込みが終了する前に次の割り込みが発生した場合（これが多重割り込み）にlr_irqの値が上書きされるため，2回目の割り込みが終了するときはよいのですが，1回目の割り込みに戻ったときにLRが割り込み前と異なる値になり，以後の処理に不ぐあいが生じます．

このあたりが割り込みを含むプログラムの難しさで，すぐにプログラムが停止するような症状ではなく，徐々にスタック（ポインタ）がおかしくなり，そのうち例外が発生するなどして問題が見つかります．すぐに問題を特定しにくいため，こういったバグの原因を探すのはたいへんです．

リアルタイムOSを使うと，こういった部分はリアルタイムOSに任せることができるため，開発効率が上がるということになります．

上記の解決法ですが，IRQ割り込み処理をIRQモード以外で行えばよいので，SVCモードで割り込み処理を実行するようにしました．**リスト8**に示すようにIRQ割り込みが入るとIRQモードになりますが，割り込み入り口でSVCモードに切り替えてからそれ以降の処理を行うことで，多重割り込みを可能にします（何重割り込みの場合でも，SVCモードに切り替えて処理する）．

● TOPPERS/JSPカーネルを移植

まず，JSPカーネルの構造ですが，大まかにターゲット

リスト8　割り込み処理時にIRQからSVCモードに切り替える

```
interrupt:
    mov        sp,  #(CPSR_MD_SVC|CPSR_IRQ|CPSR_FIQ)
    msr        cpsr_all,  sp                    /* Supervisor mode */
    stmfd      sp!,  {r0-r3,ip,lr,pc}               /* GPR -> stack */
    mov        r0,  #(CPSR_MD_IRQ|CPSR_IRQ|CPSR_FIQ)
    msr        cpsr_all,  r0                           /* IRQ mode */
    sub        r1,  lr,  #4                /* return address -> r1 */
    mrs        r2,  spsr_all
    and        r0,  r2,  #CPSR_FIQ
    orr        r0,  r0,  #(CPSR_IRQ|CPSR_MD_SVC)
    msr        cpsr_all,  r0                    /* Supervisor mode */

    str        r1,  [sp,#0x18]             /* over-write pc (<- r0) */
    stmfd      sp!,  {r2}                  /* r0(SPSR_IRQ) -> stack */
     ・・・
```

IRQモードなのでsp_irqを使い，ほかのレジスタを汚さずに一度SVCモードに切り替える

スクラッチ・レジスタとLR，PCをスタック（sp_svc）に保存

割り込みからの戻り先（LR_irq-4）とSPSR_irqをスタックに保存するため，IRQモードに

割り込み前のFIQの設定を残しつつ，多重割り込みのためSVCモードへの切り替えとIRQ割り込みの禁止

前に保存したPCに割り込みからの戻り先（r1）を上書きし，SPSR_irq（r2）をスタックに保存．その後，SVCのまま割り込み処理を行う

3

リスト9 割り込みベクタの処理

```
void  sys_initialize() {
    int i;
    init_clock();
    init_gpio();
    init_IRQ();
  /*
   * SRAM
   */
    for(i = 0; i <=7; i++){
        arm_vector_add[i] = arm_vector_add[i] + VCT_TB_SRAM/4;
    }
    for(i=0; i <=14; i++) {
         *((volatile UW *)VCT_TB_SRAM + i) = *((volatile UW *)VCT_TB + i);
    }
     sil_wrw_mem((VP)REMAP, REMAP_SRAM);
    /*
     *  ARMのベクタ・テーブルへ割り込みハンドラを登録
     */
    arm_install_handler(IRQ_Number, IRQ_Handler);

    . . . .
}
```

arm_vector_add は，0番地からの例外ベクタ・テーブルのアドレスを持っているので，これにSRAMの先頭番地を付加し，SRAMに配置する例外ベクタ・テーブルのアドレスにする

SRAM領域の先頭に，Flash/EE領域のベクタ・テーブルと各例外ハンドラのアドレス部分をコピーする

REMAPレジスタのREMAPビットをセットすることで，SRAM領域が0番地以降にリマップする．これで，例外時はSRAM領域に作成したベクタ・テーブルを参照する

IRQ_Numberが対応する例外のハンドラのアドレスに，IRQ_Handlerを上書きする．ここでは，SRAM領域のベクタ・テーブルを書き換えることになる

CPU依存部と非依存部に分かれます．移植にあたり，コーディングが必要なのは依存部です．その依存部も，プロセッサ依存部とシステム依存部に分かれています．その分かれかたは，プロセッサによって異なっています．SH-3プロセッサではシステム依存部はシリアルのドライバのみですが，ARMプロセッサでは割り込み出入口処理やシリアル通信，タイマのドライバなども含まれています．どのターゲットに移植するかで，作業の手間が変わってきます．

今回の移植対象は本誌2006年3月号付属のCPU基板で，プロセッサ・コアはARM7TDMIです．TOPPERS/JSPカーネル(1.4.2)がサポートする既存のARMアーキテクチャはARMv4，システム(ボード)はIntegratorとなっています．そのソース・コードを見て，config/armv4直下のファイルはそのまま使えると判断しました．そして，config/armv4/frk_aducというディレクトリを作成し，システム(ボード)に依存するコードをここに入れることにしました．

さて，そのシステムに依存するコードを一から作ってもよいのですが，同じARMのIntegrator(config/armv4/integrator)のシステム依存部のコードを参考にすることにしました．基本的にはシステム依存にある関数を移植対象のシステム用に変更すれば移植は完了します．

システム部分のみの移植なので，ミスが発生しやすいディスパッチャやその周りのコードは書く必要がありませんでした．割り込みも多重割り込みのプログラム作成を行ったときにTOPPERS/JSPカーネルを意識して書いていたので，移植はスムーズに行うことができました．ただ1点，例外ベクタのところでは失敗してしまいました．

その失敗とは，TOPPERS/JSPカーネルのarmv4コードでは例外ベクタを書き換えるのですが，その例外ベクタをFlash/EE領域に配置していたのです．Flash/EE領域に書き込むにはある手順が必要になるのですが，それを無視すると例外ベクタを設定するところで例外が発生するというヘマをやってしまいました．最初は，テーブル書き換えを行わない方向で考えていたのですが，それだとconfig/armv4直下のファイルを書き換えることになってしまいます．当初の「差分で作る」という目標から，**リスト9**に示すような容量の少ないSRAMに例外ベクタ・テーブルの領域を確保・作成し，SRAM領域をリマップするようにしました．

● サンプル・プログラムを動かす手順

次に，本稿で紹介したサンプル・プログラム(本誌付属のCD-ROMに収録されている)を動かす手順について述べます．

サンプルの各ディレクトリ(asm_led，c_led，timer_int，uart，multi_int，jsp/program/sample1)にARMWSDでフラッシュ・メモリに書き込める形式のファイル(拡張子がhex)があります．ARM7基板とパソコンをシリアル・ケーブルで接続して，このファイルをARMWSDツールを利用して書き込みます．

実行する際に，uart，multi_int，JSPのサンプル・プログラムでは，Teratermなどのシリアル・ポートに接続できるソフトウェアを使用します．通信は，ボーレート9600 bps，データ長8ビット，パリティなし，ストップ・ビット1ビット，フロー制御なしと設定とします．ARMWSDで書き込む場合，ほかのソフトウェア(例えばTeratermなど)が同じシリアル・ポートを使用していると書き込みで

リスト10
スタック・サイズを変更
SRAM領域が少ないので，スタック・サイズを小さくしておいた．

```
#elif defined(ARMV4)

#define CPUEXC1                       4         /* ロード・エラー例外 */
#define RAISE_CPU_EXCEPTION           (*((volatile int *) 0x0007FFF0))
#define         STACK_SIZE            512       /* タスクのスタック・サイズ */
```

きません．書き込む場合は，ほかのソフトウェアのシリアル・ポート接続は切断しておいてください．

JSP以外のサンプルを変更（例えば改良）した場合，ファイル名がそのままであれば，

```
> make
```

と入力するだけで，コンパイルからARMWSD対応のhexファイルの作成までが完了します（Makefile利用時）．ファイル名などを変更する場合は，Makefileも変更する必要があります．

● TOPPERS/JSPカーネルを動かす手順

では，TOPPERS/JSPカーネルを動かしてみましょう．まず，TOPPERS/JSPカーネル本体（1.4.2，1.4.1でも可）を用意します．初めて使うときは，ソース・ツリーのcfg/でmake depend，makeを実行して，chk.exe，cfg.exeを作成する必要があります．TOPPERS/JSPカーネルのソース・ツリーのconfig/以下に今回作成したarmv4ディレクトリをコピーします（arm4以下のcpu_support.Sに変更すべき点があり，JSPカーネルの次回リリースであるcpu_support.Sを，許可を得て使用しているため）．

JSPカーネルを使用してプログラムを作成する場合は，メインのC言語ファイルとCFGファイルの名まえを同じにします．例えば，C言語ファイルとしてhoge.c，CFGファイルとしてhoge.cfgとします．この二つのファイルを適当なフォルダに置き，configureを実行します．

```
> ../configure -C armv4 -S frk_aduc -A hoge
```

これにより，Makefileが生成されます（configureのパスは，適宜変える）．

あとは，以下のコマンドを実行すると，jsp.hexというARMWSDで書き込み可能なファイルができ上がります．

```
> make depend
> make
> arm-elf-objcopy -O ihex -S jsp.exe jsp.hex
```

今回は，TOPPERS/JSPのサンプルをjsp/program/sample1のフォルダに置いています．使用法はsample1.cのコメント部を参考にしてください．今回のARM7基板はSRAM領域の容量が小さいため，**リスト10**のようにsample1.hにおいてスタックのサイズを小さめにしています．少し大きなプログラムを動作させるには，なんらかの形でメモリを増設する必要があると思います．

今回移植したTOPPERS/JSPは公開しますので，ぜひ多くの方にリアルタイムOSを用いたソフトウェア開発に触れていただきたいと思います．また，最近はいろいろなものがブラックボックス化されて内部が見えないものが多いのですが，ソース・コードが公開されているTOPPERSを見ることは，たいへん貴重な経験になると思います．

3

参考・引用*文献

(1) 高田広章，岸田昌巳，宿口雅弘，南角茂樹；リアルタイムOSと組み込み技術の基礎―実践μITRONプログラミング，CQ出版社，2003年7月．
(2) ITRONホームページ，http://www.ertl.jp/ITRON/home-j.html
(3) TOPPERSプロジェクトのホームページ，http://www.toppers.jp/
(4) GNUWingのホームページ，http://www.embedded.jp/gnuwing/index.html

なかの・としかず
ひらお・ともや
あかぼし・ひろき
（株）ロジック・リサーチ

<筆者プロフィール>
中野俊和． フルカスタム・チップの設計，RTL設計，リアルタイムOSの移植までが業務範囲．TOPPERSをこよなく愛する若手技術者．SSEST2 実行委員．
平尾智也． HDLを書けるソフトウェア技術者として，ソフトウェアからハードウェアまで手広く手がけるユーティリティ・パーソン．ハードウェアとソフトウェアのはざまに生息しながらも，開発手法としてのソフトウェア工学に注目中．
赤星博輝． ハードウェアからソフトウェアまで担当するなんでも屋．現在はディジタル検証ではVerification Methodology ManualとSystemVerilogを推進し，ハードウェア検証のためにリアルタイムOS技術にも進出．